Escuela Técnica Superior de Ingeniería Informática
Grado en Ingeniería del Software

Sistema web para la ejecución remota de Maude
Web system for remote execution of Maude

Realizado por

FRANCISCO JAVIER MARTÍN CUENCA

Dirigido por

DR. JOSÉ MARÍA ALVAREZ PALOMO

Y

ANTONIO MANUEL MORENO DELGADO

Departamento

LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

UNIVERSIDAD DE MÁLAGA

Málaga, Julio 2017.

Fecha de la defensa:

El Secretario del Tribunal

Resumen

El objetivo del presente trabajo de fin de grado consiste en la elaboración de un sistema web que permita al usuario utilizar el sistema Maude sin necesidad de instalar ningún software adicional.

Esta necesidad surge a raíz de la dificultad de instalar el entorno Maude en sistemas Windows. Gracias al software desarrollado, es posible acceder a un terminal que opera este interprete por medio de un navegador web.

El proyecto se desarrollará utilizando orientación a eventos con el lenguaje de programación JavaScript tanto para el cliente como para el servidor.

Palabras claves: Terminal web, Maude, Node.js, WebSocket, JavaScript

Abstract

The objective of this final degree project consists of the elaboration of a web system that allow user to use a Maude enviorment whitout install any additional software.

This need arises from the difficulty to install the Maude enviorment in Windows systems. Thanks to the developed software, it's possible access to a terminal that operate this interpreter in a web browser.

The project will be developed using the event oriented paradigm, with JavaScript programing language, for the client side, and the server side.

Keywords: Web terminal, Maude, Node.js, WebSocket, JavaScript

Índice general

Capítulo 1

Introducción

La ingeniería dirigida por modelos es una metodología de desarrollo software que se centra en la creación y explotación de modelos de dominio, más que en algoritmos. Un modelo de dominio es una representación abstracta de los conocimientos y actividades que rigen el dominio de una aplicación particular. Esta metodología busca aumentar la productividad mediante la maximización de la compatibilidad entre los sistemas, a través de modelos estandarizados, simplificando el proceso de diseño mediante patrones de diseño.

1.1. Motivación

La ingeniería dirigida por modelos precisa de herramientas para la creación de modelos de dominios, definiendo conjuntos de datos y operaciones sobre ellas. Para ello se presenta Maude, un lenguaje e interprete para especificaciones formales mediante el uso de términos algebraicos. Este interprete permite la verificación de propiedades y transformaciones sobre modelos y permite ejecutar la especificación como si fuera un prototipo.

Maude permite su instalación en entornos UNIX, no obstante no es compatible en entornos Windows. Para permitir un mejor acceso al lenguaje surge la idea que motiva este Trabajo de Fin de Grado, una consola web para la ejecución online de Maude.

1.2. Objetivos del Trabajo de Fin de Grado

El objetivo del Trabajo de Fin de Grado es la creación de un entorno online que permita la ejecución de instrucciones Maude. La aplicación cuenta con las funcionalidades de una consola nativa recreadas de forma telemática, tales como historial de comandos o carga de módulos Maude.

1.3. Herramientas empleadas

1.3.1. Lenguajes

HTML5

HTML son las siglas de HyperText Markup Language (Lenguaje de Marcado de Hipertexto). Es el lenguaje sobre el que se construye la web y de su desarrollo se encarga la W3C. No es un lenguaje de programación, sino de marcado. Permite estructurar la información que se le presenta al usuario mediante el uso de etiquetas ya preestablecidas.

En su quinta revisión agrega nuevas etiquetas y añade nuevas funcionalidades nativas como reproductor de video y audio. Google penaliza las páginas que empleen versiones anteriores para el SEO en su motor de búsqueda.

Actualmente es posible definir tus propias etiquetas mediante el uso de frameworks JavaScript como Angular y la arquitectura orientada a componentes, aunque aun no es una característica propia del lenguaje.

CSS3

CSS son las siglas de Cascade Style Sheet (hoja de estilo en cascada en español) es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado. Es usado para establecer el diseño visual de las páginas web, e interfaces de usuario escritas en HTML. Está diseñado principalmente para marcar la separación del contenido del documento y la forma de presentación de este. Las características que pueden ser modificadas con CSS son por ejemplo la separación entre elementos, el tamaño, los colores o las fuentes. Esta separación permite mantener de forma mas sencilla la reutilización de estilo en distintos documentos y reducir su complejidad.

La separación del formato de presentación y el contenido hace posible presentar el mismo documento en diferentes estilos para diferentes métodos de renderizado, como para pantallas de distintos tamaños o dispositivos móviles.

Se denomina en cascada debido al orden que se establece entre sus reglas si mas de una regla se aplica a un mismo elemento, haciendo predecible que regla se aplicará en tiempo de ejecución.

JavaScript

Lenguaje de programación utilizado tradicionalmente para interactuar con HTML y dar interactividad a la web. Actualmente se puede utilizar además en el lado del servidor gracias a la plataforma Node.js. Permite manipular el árbol DOM, una estructura con forma de árbol que contiene la jerarquía de nodos que conforman el documento HTML, agregando, modificando o eliminando nodos de él.

Es un lenguaje dinámico y débilmente tipado con lo que denominan “duck-type”, donde el conjunto actual de métodos y propiedades determina la validez semántica, en lugar de la herencia de una clase en particular o la implementación de una interfaz específica. Otra característica propia del lenguaje es su orientación a eventos. Durante la mayor parte de la ejecución, el proceso se encuentra dormido y a la espera de que se produzca algún evento. Cuando este sucede, se captura y se ejecuta la función asociada a ese evento.

1.3.2. Tecnologías

Node.js

Definido en su página web como “Un entorno de ejecución para JavaScript con el motor V8 de Chrome, Node.js es orientado a eventos, no bloqueante en entrada y salida haciéndolo ligero y eficiente.”. Su principal cualidad es que permite utilizar JavaScript en el lado del servidor.

Emplea un paradigma diferente al tradicional orientado a hilos como el empleado por Apache. Este cambio de paradigma supone grandes oportunidades para la plataforma Node.js. La orientación a eventos le permite en un único hilo de ejecución (con el consiguiente ahorro en cambios de contexto) atender todas las peticiones que se le hagan. Es ampliamente utilizado en sistemas para comunicaciones en tiempo real.

El modelo de entrada y salida no bloqueante le permite atender todas las peticiones entrantes aunque aun no haya resuelto las existentes.

Su gestor de paquetes, npm, es ampliamente utilizado por distintos frameworks y bibliotecas JavaScript para el lado del cliente, con lo que su uso no se restringe solo al servidor sino que puede abarcar toda la aplicación.

JQuery

Se trata de la biblioteca JavaScript mas utilizada. Permite simplificar la forma de interactuar con los documentos html, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción mediante AJAX. Su popularidad se debe principalmente a dos factores, por un lado permite escribir funcionalidades que de otro modo requerirían de mucho más código y por permitir la compatibilidad entre los distintos navegadores.

WebSocket

Es una tecnología que proporciona un canal de comunicaciones bidireccional y full-duplex sobre un único socket TCP. Esta diseñado para ser usado en navegadores y servidores web, pero es posible su utilización en cualquier aplicación cliente/servidor. Se encuentra estandarizado por la W3C.

Debido a que las conexiones TCP comunes con puertos diferentes al 80 son habitualmente bloqueadas por los administradores de redes, el uso de esta tecnología proporciona una solución a este tipo de limitaciones, multiplexando diferentes servicios WebSocket sobre un único puerto TCP, a costa de una pequeña sobrecarga del protocolo.

Express.js

express

Es una framework para la construcción de aplicaciones web para la plataforma Node.js. Esta diseñado para la creación de aplicaciones web y APIs. Es el estándar de facto para Node.js cuando se quieren crear aplicaciones web. Forma junto con MongoDB, Angular y el propio Node.js el conocido stack MEAN.

1.3.3. Bibliotecas especificas

Socket.io

Es una biblioteca para aplicaciones web en tiempo real. Permite comunicación bidireccional entre los clientes y el servidor. Dispone de dos partes casi idénticas, una para la parte del cliente y otra para la parte del servidor. Como Node.js está orientado a eventos.

Utiliza el protocolo WebSocket, pero no es solo una envoltura para dicho protocolo. Añade varias características como difusión a múltiples sockets, almacenamiento de datos a cada cliente y entrada y salida asíncronas.

JQuery Terminal Emulator

Biblioteca de código libre que emula un terminal en el navegador. Permite configurarla para adaptar el prompt o autenticación de usuario. Incluye características propias de una consola nativa como un historial de comandos o auto compleción de comandos con el tabulador. Solo se ocupa de recoger los comandos y mostrar texto por pantalla, dejando al desarrollador la tarea de evaluar los mismos o la comunicación con el servidor.

Capítulo 2

Especificación

2.1. Requisitos funcionales

Estos requisitos pretenden describir las funcionalidades de una consola en modo local, pues son las funcionalidades que se pretenden emular en el software a desarrollar. La primera versión solo incluía los dos primeros, pues pueden considerarse como los únicos imprescindibles. Los demás pretenden aumentar la sensación de estar frente a una consola nativa. El último requisito se agregó para ofrecer ayuda al usuario sobre Maude.

- **Introducción de comandos**
  El usuario podrá introducir comandos Maude en la terminal, y obtener la respuesta impresa en la consola, tal como lo haría en una consola nativa.

- **El usuario puede subir módulos.**
  El usuario puede subir sus propios módulos mediante la correspondiente entrada. Deben ser archivos de texto plano.

- **Historial de comandos.**
  La aplicación dispondrá de historial de comandos, en los que el usuario podrá navegar con las teclas de dirección por los comandos introducidos anteriormente.

- **Borrado de entrada aun no introducida.**
  La interfaz permitirá que el usuario pueda borrar la entrada actual si aun no ha sido introducida pulsando la tecla de borrado.

- **Reinicio de consola.**
  El usuario podrá reiniciar la consola, y con ello el proceso Maude. Si introduce el comando `quit` o se pulsa la combinación de teclas `Ctrl+C` reiniciará la consola.

- **Enlaces de interés**
  El usuario dispondrá de enlaces sobre temas relacionados con Maude.

2.2. Requisitos no funcionales

Los requisitos no funcionales describen aquellas características que no ofrecen funcionalidades, pero son necesarias para definir el sistema. Ambos se agregaron después de realizar el primer prototipo. Tras definir varias necesidades con los directores, se redactó la siguiente lista.

- **Tiempo de ejecución limitado**
  Los ejecución de un comando no podrá superar un determinado umbral de tiempo. Si se supera se reiniciará la consola.

- **Comandos restringidos.**
  La aplicación bloqueará comandos ajenos a Maude. Se considerarán comandos ajenos a Maude aquellos comandos que creen un proceso diferente como pueden ser los instaladores o que interactúen con la aplicación servidor. También bloqueará aquellos comandos que precisen de directorios o archivos, tales como `load`, `pwd`, `cd` y similares.

2.3. Casos de uso

A continuación se mostrará una descripción más en profundidad de las interacciones del usuario con el software por medio de la descripción de los casos de uso.

1. **Introducción de comandos**

   a) Contexto de uso: Cuando el usuario quiere introducir un comando Maude.

   b) Nivel: Usuario.

c) Participantes y objetivos:

Usuario: Ejecutar un comando Maude.

d) Precondiciones:

Debe tener seleccionada la terminal.

Debe haber conexión con el servidor.

e) Garantías de éxito: Se envía el comando al servidor y se visualiza la respuesta.

f) Escenario de éxito principal

1) El usuario escribe el comando.
2) El usuario pulsa la tecla `intro`.
3) El terminal valida el comando.
4) El terminal envía el comando al servidor.
5) El servidor redirige el comando recibido al proceso Maude.
6) El proceso Maude emite su salida por la salida estándar.
7) El servidor emite al respuesta del proceso por la salida estándar.
8) El terminal imprime la respuesta recibida del servidor.

g) Escenarios alternativos

1) Si el comando introducido no es valido, como pueden ser los que empiecen por `load`, `cd` o similares que el motor de Maude si admite, pero por seguridad de la aplicación se han bloqueado, se informa al usuario a través de la salida de error.
2) En cualquier momento si el usuario introduce un comando que empiece por `quit` o pulsa `Ctrl+C` se recarga la página y con ello se elimina forzosamente el proceso Maude en el servidor.

2.3. Casos de uso

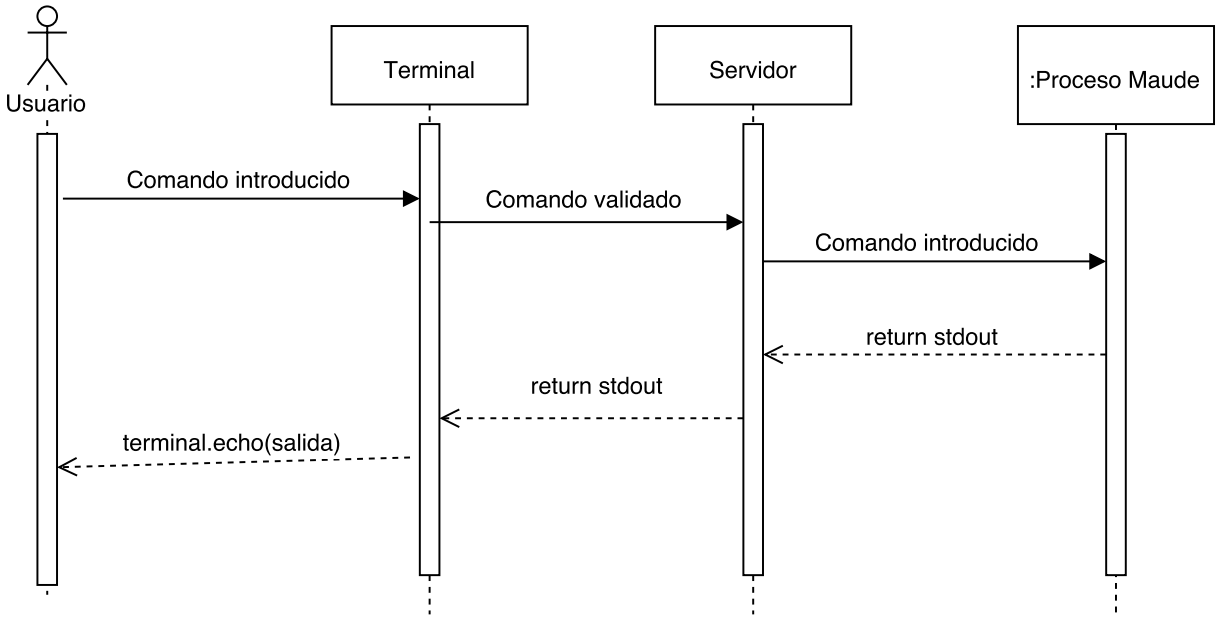

Figura 2.1: Introducción y salida de una comando

2. **Subida de módulos**

   a) Contexto de uso: Cuando el usuario quiere introducir varios comando Maude.

   b) Nivel: Usuario.

   c) Participantes y objetivos:

   Usuario: Ejecutar varios comando Maude.

   d) Precondiciones:

   Debe haber establecido la conexión con el socket.

   e) Garantías de éxito: Se envían los comando al servidor y se visualiza la respuesta de cada comando.

   f) Escenario de éxito principal

   1) El usuario pulsa sobre el enlace correspondiente.
   2) El usuario selecciona un fichero.
   3) El terminal envía lee el fichero.
   4) El terminal envía cada comando al servidor.
   5) El terminal imprime la respuesta a cada comando recibida desde el servidor.

   g) Escenarios alternativos

1) Si el fichero no es de texto plano, no es valido se emite una alerta informando al usuario de este hecho.

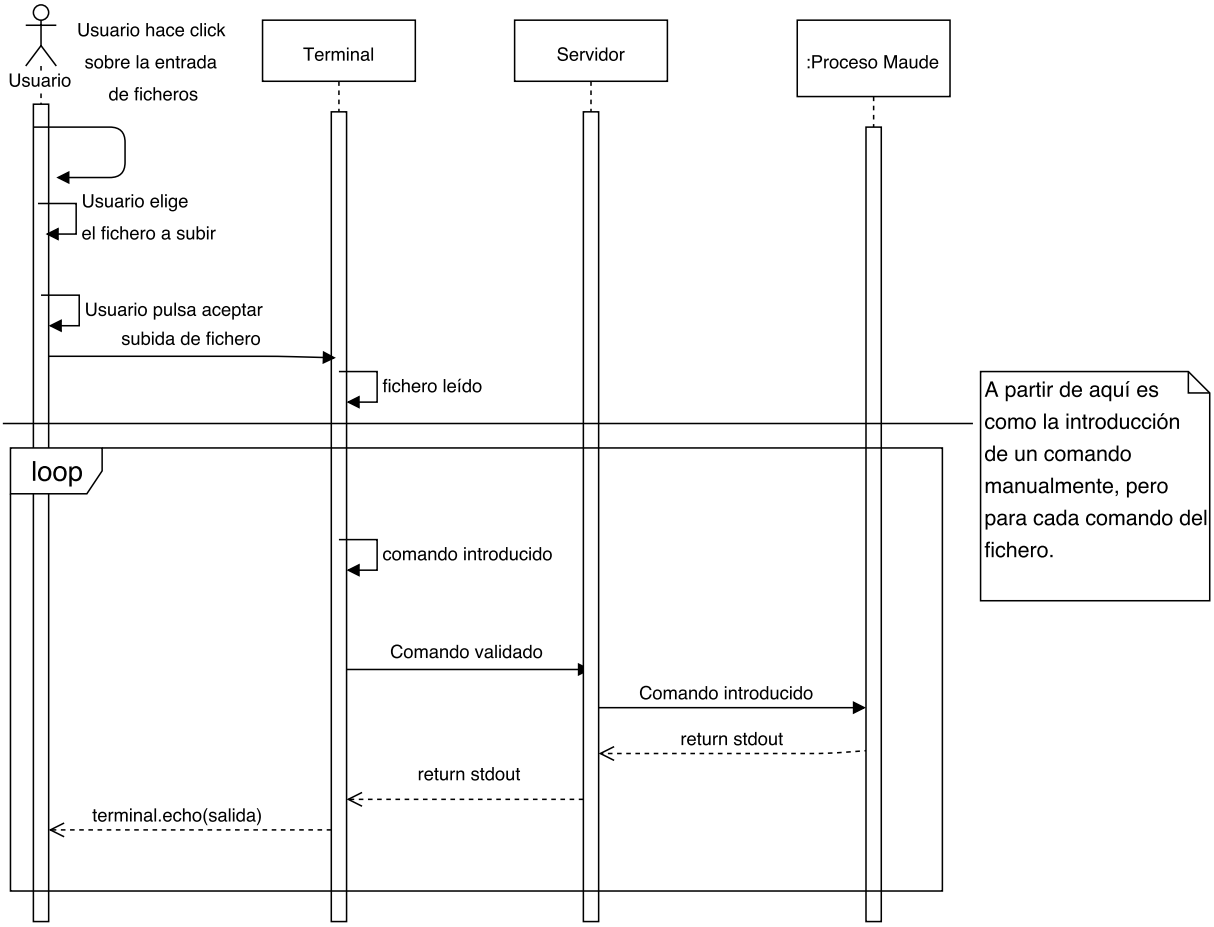

Figura 2.2: Subida de un módulo

3. **Navegación por el historial de comandos**

a) Contexto de uso: Cuando el usuario quiere introducir un comando anteriormente escrito.

b) Nivel: Usuario.

c) Participantes y objetivos:

Usuario: Ejecutar un comando anteriormente escrito.

d) Precondiciones:

Debe haber establecido la conexión con el socket.

Debe haber introducido algún comando anteriormente.

e) Garantías de éxito: Se accede a comandos anteriormente escritos.

f) Escenario de éxito principal

1) El usuario pulsa sobre la flechas arriba del teclado.
2) El terminal recupera el comando previamente introducido.
3) El terminal muestra el comando previamente introducido en la línea actual.

g) Escenarios alternativos

1) Si el usuario pulsa la tecla de flecha abajo podrá acceder a comandos anteriormente introducidos, pero posteriores al actualmente seleccionado.

4. **Borrado de una entrada aun no introducida**

a) Contexto de uso: Cuando el usuario quiere borrar los caracteres de la línea actual.

b) Nivel: Usuario.

c) Participantes y objetivos:

Usuario: Borrar caracteres de la línea actual.

d) Precondiciones:

Debe tener la terminal seleccionada.

e) Garantías de éxito: Se borrar los caracteres anteriores a la posición del cursor.

f) Escenario de éxito principal

1) El usuario pulsa la tecla de borrado del teclado.
2) El terminal elimina el carácter situado a la izquierda del cursor.

g) Escenarios alternativos

1) Si el cursor se encuentra al inicio de la línea, no hay caracteres a la izquierda, por tanto no se borra ningún carácter.

5. **Reinicio de consola**

a) Contexto de uso: El usuario quiere poner fin al proceso Maude actual.

b) Nivel: Usuario.

c) Participantes y objetivos:

Usuario: Eliminar la instancia actual.

d) Precondiciones:

Debe estar mostrada la terminal.

Debe haber conexión con el servidor.

e) Garantías de éxito: Se ha creado otra instancia Maude en se servidor y los comandos siguientes se dirigirán a ella.

f) Escenario de éxito principal

1) El usuario recarga la página.
2) El servidor elimina el proceso Maude asociado y cierra la conexión.
3) El terminal se vuelve a crear y se conecta al servidor.
4) El servidor acepta la conexión y crea un nuevo proceso Maude al que le pasará las entradas de usuario.

g) Escenarios alternativos

1) Se consigue el mismo efecto pulsando la combinación de teclas `Ctrl+C`.
2) Si se interrumpe la conexión durante el proceso, no es posible crear la nueva instancia Maude.

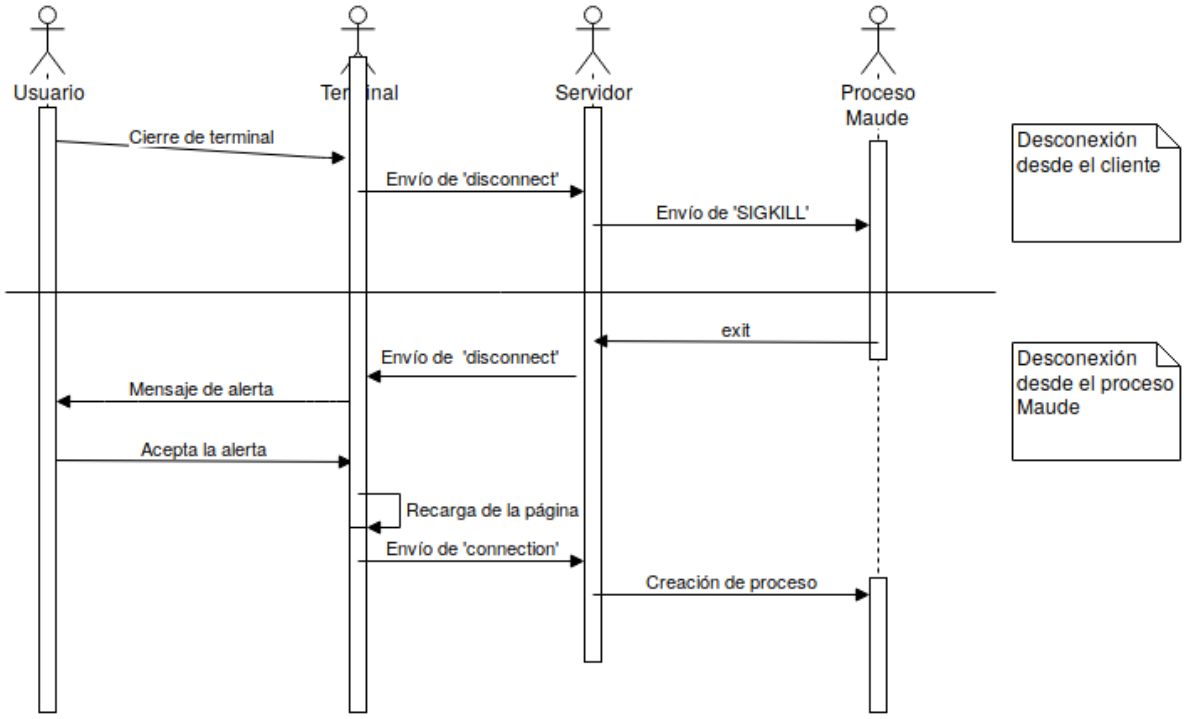

Figura 2.3: Desconexión desde el servidor y desde el cliente

6. **Acceso a enlaces de interés**

a) Contexto de uso: El usuario quiere acceder a información sobre el lenguaje Maude.

b) Nivel: Usuario.

c) Participantes y objetivos:

Usuario: Obtener información.

d) Precondiciones:

Debe haber conexión a Internet.

e) Garantías de éxito: El usuario accede a documentación sobre el lenguaje Maude.

f) Escenario de éxito principal

1) El usuario pulsa sobre alguno de los enlaces.
2) El navegador abre una nueva pestaña con el contenido del enlace seleccionado.

g) Escenarios alternativos

1) El usuario selecciona la opción de abrir el enlace en una nueva ventana.
2) El usuario selecciona la opción de abrir el enlace en una nueva pestaña.

Capítulo 3

Diseño

3.1. Diseño de interfaz

Desde la aparición de los teléfonos inteligentes, las visitas a sitios web desde dispositivos móviles ha aumentado enormemente, hasta hoy día en el que la mayor parte del tráfico web proviene de estos dispositivos. Con esto surgen nuevas tendencias en el desarrollo web como son Responsive design y Mobile first. Responsive design consiste en adaptar la interfaz de la pantalla de forma que no sea necesario hacer scrolling horizontal para ver todo el contenido que la página ofrece. Mobile first consiste en diseñar la página para dispositivos móviles y luego adaptarla a pantallas mas grandes. En un desarrollo web para un público más genérico, se seguiría un enfoque Mobile first para el diseño de la interfaz, pero no sucede así con esta aplicación.

El público objetivo de esta aplicación es de un perfil técnico, acostumbrado a usar ordenadores (hacemos distinción entre ordenadores y teléfonos inteligentes), los cuales accederán a la aplicación desde un dispositivo con pantalla grande (entendamos pantalla grande a aquella con un ancho superior a 1000 pixeles). Dado este público objetivo se ha diseñado todo pensando en una versión de escritorio, y luego se ha adaptado a versión móvil.

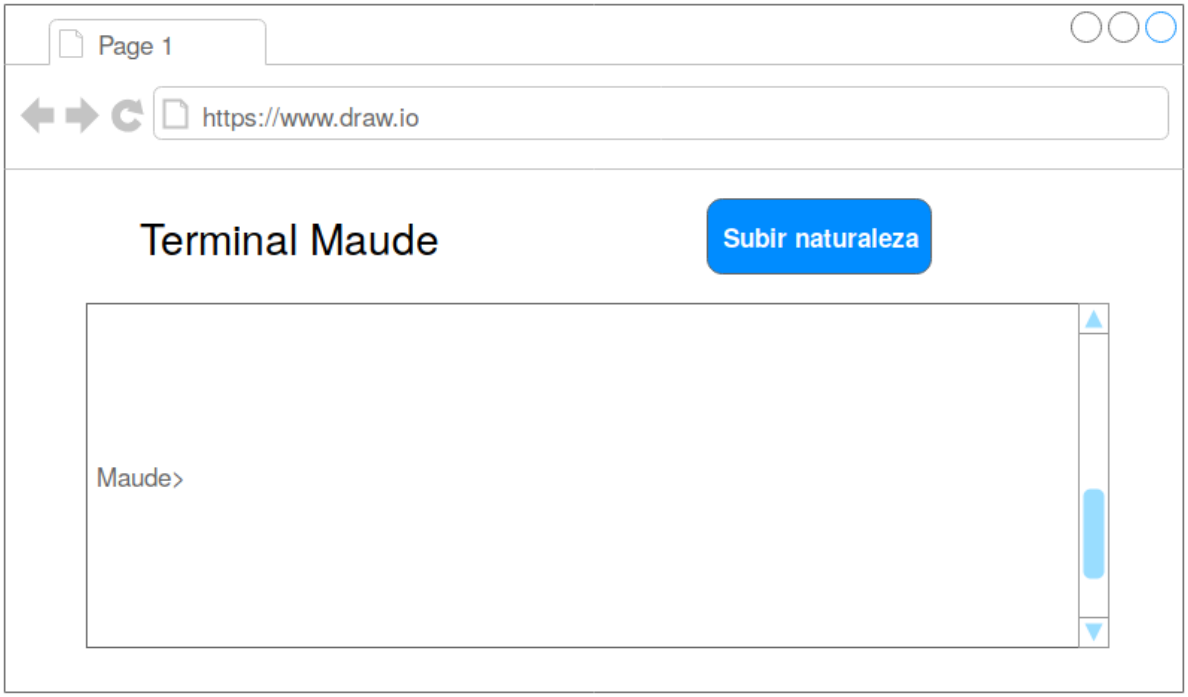

Figura 3.1: Mockup original del proyecto

Mockup de la interfaz Este es el primer diseño de la interfaz de la aplicación. Durante el proceso de creación de la interfaz gráfica se barajaron varias ideas referentes a como permitir al usuario la subida de módulos. Una de ellas fue la inclusión de un panel de texto en el lateral de la terminal y que el usuario escribiese ahí los módulos para mandarlos al servidor cuando escribiese el comando `load`. Finalmente se deshecho esta posibilidad para evitar posibles problemas con la biblioteca usada para emular el terminal. Esta biblioteca funciona capturando las pulsaciones de teclas y a veces puede suponer una interferencia cuando hay campos de texto en los que el usuario deba escribir. La disposición de los elementos visuales externos al terminal como los enlaces y el título también se han visto afectado durante el proceso de desarrollo.

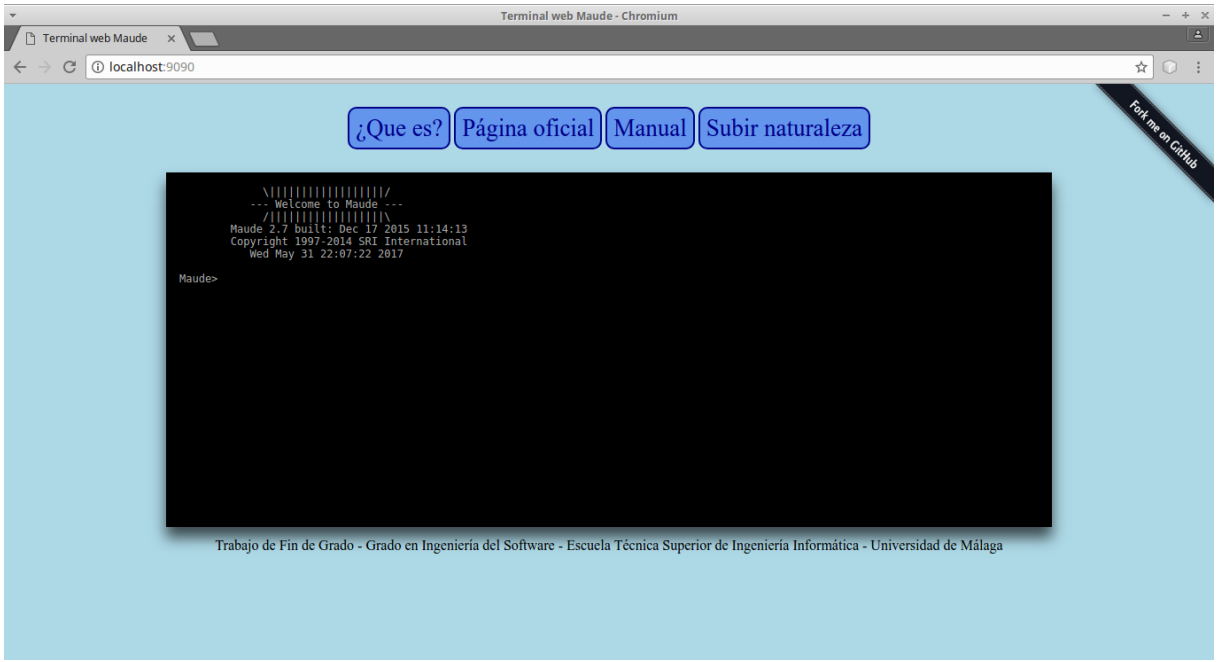

Figura 3.2: Diseño final de la interfaz de escritorio

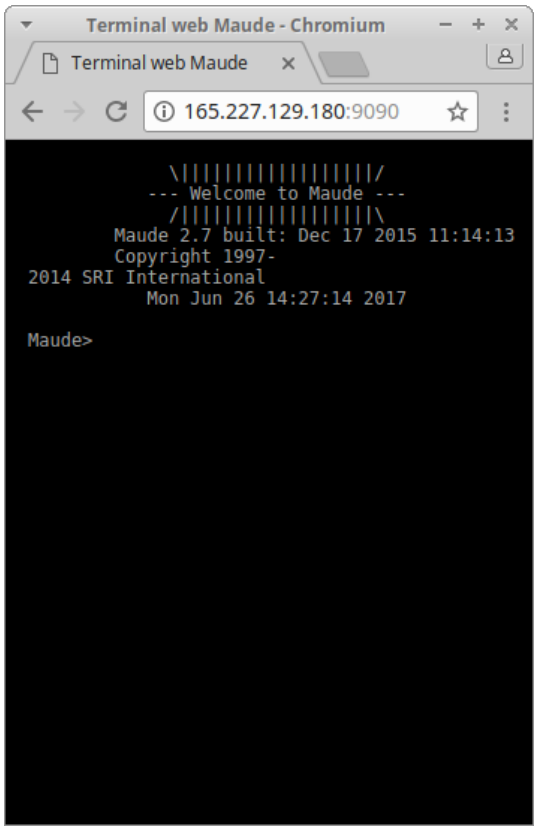

Figura 3.3: Diseño final de la interfaz móvil

Diseño final Finalmente este es el aspecto final de la aplicación tanto en versión de escritorio como en versión móvil. La parte superior de la pantalla dispone de un menú con los enlaces que puedan ser de interés para el usuario tales como el manual de Maude.

La entrada de ficheros para la subida de módulos se encuentra disponible en ese mismo menú. Visualmente se muestra como un enlace de forma que el botón original se encuentra oculto y no es posible pulsarlo directamente. Esto se hace para mantener una coherencia estética entre los elementos de la interfaz.

En los dispositivos móviles la pantalla es mucho menor que en equipos de escritorio, por tanto se han eliminado todos los elementos externos al terminal para maximizar el espacio disponible. Gracias al uso de la propiedad breaklines de CSS se evita el scrolling lateral.

3.1. Diseño de interfaz

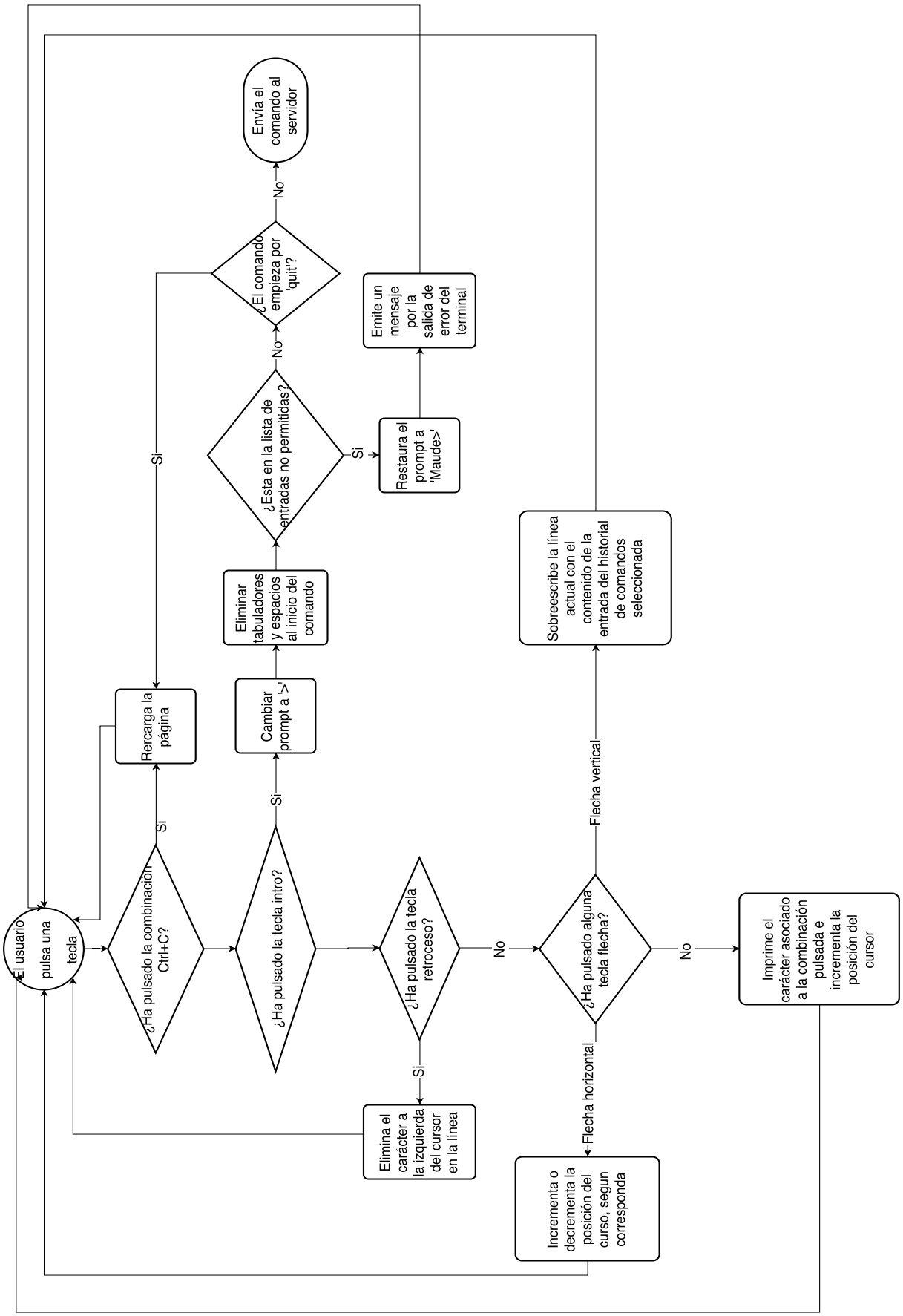

Figura 3.4: Funcionamiento del emulador de terminal

Funcionamiento del terminal Durante las investigaciones para averiguar como construir un emulador de terminal en el navegador se encontraron varias alternativas. Algunas presentaban un funcionamiento muy limitado, permitiendo únicamente introducir comandos (el borrado no era posible), otros eran muy difíciles de adaptar, con un diseño de código ininteligible.

Los emuladores de terminal mas robustos utilizaban la biblioteca JQuery para su funcionamiento, con un esquema similar al siguiente, aunque con pequeñas variaciones de una versión a otra.

Cabe destacar que estos emuladores no disponen de un campo de entrada de texto usual, donde el usuario introduce el comando. El terminal registra cada pulsación de teclado (o combinación de teclas) e imprime en la línea actual la tecla pulsada (o combinación si procede, como en el caso de las mayúsculas).

Dispone de un cursor virtual, cuya posición es almacenada en una variable interna de la biblioteca. La visualización del cursor se consigue mediante el uso de JavaScript para ejecutar una función periódica que muestra el cursor y otra que lo oculta tras el mismo intervalo de tiempo logrando un efecto de parpadeo.

El contenido de la línea actual es almacenado cuando el usuario presiona la tecla `Intro`. Cuando el usuario presiona las teclas de flecha arriba o abajo, se borra el contenido de la línea actual y es reemplazado por el contenido de una línea introducida anteriormente, según la navegación del usuario por el historial de comandos. Si no se elimina la caché del navegador, el historial de comandos se almacena de una conexión a otra, del mismo modo que en una consola nativa.

3.2. Decisiones de diseño

Durante esta sección se hablará de los problemas encontrados durante la realización del TFG, las alternativas estudiadas para resolverlos y las decisiones tomadas.

3.2.1. Tiempo de ejecución limitado para los comandos

En Maude una reducción puede entrar en un bucle infinito y bloquear todo el proceso. Para remediar esto se pensó en limitar el tiempo de ejecución de un comando.

En JavaScript existe una función denominada `setTimeout(function,ms)` que ejecuta la función `function` que se le pasa como parámetro transcurridos `ms` milisegundos. En la parte del servidor se usará para ejecutar una rutina que elimina el proceso

Maude tras un periodo de tiempo. A esta función nos referiremos como temporizador por cuestiones de comodidad.

Las primeras versiones limitaban el tiempo con un temporizador que se iniciaba cuando se introducía el comando en el proceso y se detenía cuando se producía la respuesta. Esto causaba problemas cuando se carga un módulo. Al no producirse la salida antes de la llegada del siguiente comando, se perdía la referencia al temporizador haciendo imposible desactivarlo. Para remediar este problema se amplió a una cola de temporizadores para permitir la ejecución de lotes de comandos. La solución seguía sin ser suficiente ya que había comandos que no emitían salida a pesar de haber terminado su ejecución, como pueden ser los comandos '`fmod NOMBRE_MODULO is`' y similares. Al no emitir salida no se activa la función que elimina un temporizador de la lista, por tanto siempre se quedaría algún temporizador sin desactivar.

Finalmente se optó por limitar el tiempo de conexión con un temporizador después de crear la conexión y el proceso Maude asociado a ella. Cuando finaliza este temporizador, se considera que el proceso ha entrado en una reducción que implica un bucle infinito y por tanto se ejecuta una rutina que elimina el proceso Maude y cierra la conexión.

3.2.2. Versión para dispositivos con pantalla táctil

La biblioteca empleada para emular un terminal de comandos aporta numerosas características de las que una consola nativa posee, pero debido a su diseño no funciona en terminales móviles que carecen de teclado físico. Según las pruebas realizadas funciona tanto en escritorio con teclado físico como en iOS con el navegador Safari. No funciona correctamente en dispositivos Android.

La consola carece de una entrada de texto como tal, su funcionamiento se basa en capturar las pulsaciones de teclado del usuario, averiguar qué tecla es la que se ha pulsado y mostrar en la línea actual el carácter asociado a dicha tecla. Si se pulsa la tecla de borrado, se produce un proceso similar, pero se elimina el carácter anterior a la posición del cursor. Solo cuando se pulsa la tecla `Intro` se produce el envío del comando al servidor.

Se ha estudiado crear una aplicación móvil híbrida y usar algún plugin empleando alguno de los frameworks existentes para ese fín como PhoneGap o Ionic 3, pero ha resultado ser infructuoso debido a que los eventos que emite el teclado al pulsar sobre él son distintos a los que produce un teclado físico como el disponible en un ordenador de escritorio.

3.2.3. Emulación de terminal en el navegador

La primera versión del proyecto empleaba el módulo pty.js de Node.js para emular un terminal en el lado del servidor. El gran problema de esta alternativa era que impedía el borrado de caracteres una vez escritos en el terminal. Se buscaron alternativas en emuladores de terminal para el lado del cliente.

Estos emuladores presentaban como característica común el hecho de que no se enviaba información al servidor hasta que el usuario pulsara la tecla `Intro`. La elegida fue Jquery terminal emulator ya que es la más configurable y que mejor se adaptaba a las características del proyecto. Otras soluciones o bien eran mas difíciles de adaptar, o no poseían características que sí posee Jquery terminal emulator.

3.2.4. Conexión al servidor

Durante los primeros compases del proyecto surgió la incógnita de cómo conectar cliente y servidor. Utilizar servicios RESTFul no era una opción factible debido a que estos son sin estado y no guardan los comandos introducidos anteriormente. Tras investigar diferentes vías de comunicación se encontraron los WebSockets.

Esta solución casaba de manera excelente con Node.js que era el principal candidato para implementar el lado del servidor. Gracias a la biblioteca Socket.io, permite un comportamiento más natural de todo el sistema con un acabado muy cercano a una consola nativa.

La utilización de WebSocket permite además una solución muy sencilla y práctica al problema de las instancias compartidas, permitiendo que cada cliente solo acceda a su proceso Maude y lo elimine al cerrar la conexión.

3.2.5. Sistema de ficheros en el servidor

Una de las características principales de la aplicación es la capacidad de subir tus propios módulos al servidor. Se denomina módulo en Maude a un fichero de texto plano con cualquier extensión, con instrucciones en lenguaje Maude del mismo modo que un script en bash. Inicialmente se planteaba poder subir el fichero completo al servidor, y formar una jerarquía de ficheros virtuales. Esta opción suponía una gran complejidad, por lo que se estimo que estaba fuera del alcance del presente TFG.

La implementación final optó por desechar la jerarquía virtual en el servidor. En su lugar se implementó manualmente la lectura de ficheros de texto plano desde el

cliente y mandar línea a línea al servidor. Esta opción es mas robusta pues permite filtrar en el cliente los comandos enviados.

Capítulo 4

Implementación

A continuación se explican los diferentes archivos de la aplicación comentando el código de los mismos

4.1. package.json

```
{
  "name": "terminal_web_maude",
  "version": "1.0.0",
  "description": "Terminal para la ejecucion online de Maude",
  "main": "server.js",
  "scripts": {
    "start": "nodemon server/server.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/FranciscoJavierMartin/
       Terminal-web-Maude.git"
```

```
14  },
15  "keywords": [
16  "Terminal",
17    "Maude"
18  ],
19  "author": "Francisco Javier Martin Cuenca",
20  "license": "GPL-3.0",
21  "bugs": {
22    "url": "https://github.com/FranciscoJavierMartin/
23      Terminal-web-Maude/issues"
24  },
25  "homepage": "https://github.com/FranciscoJavierMartin/
26      Terminal-web-Maude#readme",
27  "dependencies": {
28    "express": "^4.15.2",
29    "socket.io": "^1.7.3"
30  },
31  "devDependencies": {
32    "nodemon": "^1.11.0"
33  }
34 }
```

El archivo `package.json` es un fichero de configuración generado por Node Package Manager (npm), el cual es el gestor de paquetes de Node.js. Este archivo es generado cuando se crea un proyecto Node con el comando `npm init`. Las partes más relevantes del mismo son:

- Las dependencias del proyecto,

  ```
  "dependencies": {
      "express": "^4.15.2",
      "socket.io": "^1.7.3"
  }
  ```

  En este caso solo se utilizan `express` para crear el servidor, y `socket.io` para las comunicaciones.

- Las dependencias de desarrollo,

```
"devDependencies": {
    "nodemon": "^1.11.0"
}
```

Estas son dependencias que solo se usarán para desarrollar la aplicación y no se instalarán en producción. Nodemon es un watcher que vigila los cambios realizados en los ficheros del servidor y recarga automáticamente la instancia del servidor con los nuevos cambios.

- Scripts

```
"scripts": {
  "start": "nodemon server/server.js",
  "test": "echo \"Error: no test specified\" && exit 1"
}
```

Son scripts definidos por el usuario, aunque algunos ya vienen por defecto. Para ejecutar alguno de ellos hay que realizar el comando `npm miScript`.

4.2. index.html

Propiedades

```
<meta name="viewport" content="width=device-width,
initial-scale=1, maximum-scale=1">
```

La función de esta línea es permitir una correcta visualización en terminales móviles e inhabilitar el zoom con los dedos para evitar que aparezca un scroll horizontal.

Carga de ficheros JavaScript

```
<script type="text/javascript" src="https://ajax.googleapis.com/
   ajax/libs/jquery/1.7.1/jquery.min.js"></script>
<script type="text/javascript" src="/js/jquery.terminal-1.4.2.js">
</script>
<script type="text/javascript"
src="http://192.168.1.101:9090/socket.io/socket.io.js"></script>
<script type="text/javascript" src="/js/main.js"></script>
```

En estas líneas de código se cargan las bibliotecas JavaScript que utilizará el cliente de la aplicación. Para acceder a JQuery se hace uso del CDN de Google que redirige la petición al servidor espejo más cercano según la localización del cliente. Para el uso de socket.io se carga el módulo desde el servidor de la aplicación usando el mismo que en el código de server.js.

Carga de ficheros de estilo

```
<link type="text/css" href="/css/normalize.css" rel="stylesheet" />
<link type="text/css" href="/css/jquery.terminal-1.4.2.css"
   rel="stylesheet" />
<link type="text/css" href="/css/estilo.css" rel="stylesheet" />
```

El orden en el que se aplican los estilos CSS empieza con los estilos definidos por el navegador, le siguen los definidos por el usuario en el navegador, y por último los definidos en las hojas de estilo de la página. Esto provoca que la misma página pueda verse con estilos diferentes según el navegador que se esté utilizando y provoque dolores de cabeza en los desarrolladores intentando descubrir el porqué de distintas vistas para la misma página. Para solucionar esta situación existe `normalize.css`, una hoja de estilo que sobrescribe todas las propiedades dejando el mismo estilo en todos los navegadores, luego las hojas definidas por el desarrollador sobrescribirán estas propiedades dotando a la página de un estilo homogeneizado en todos los navegadores. El fichero `jquery.terminal-1.4.2.css` contiene los estilos que se aplican unicamente a los componentes de la terminal y `estilo.css` los correspondientes a la terminal en si misma y el resto de la página.

Menu

```
<div id="menu">
<a target="_blank" class="boton"
    href="https://es.wikipedia.org/wiki/
        Maude_(lenguaje_de_programaci%C3%B3n)">
    ¿Que es?</a>

<a target="_blank" class="boton"
    href="http://maude.cs.illinois.edu/w/
        index.php?title=The_Maude_System">
```

```
    Página oficial</a>

<a target="_blank" class="boton"
    href="http://maude.cs.illinois.edu/w/images/e/e0/
        Maude-2.7.1-manual.pdf">
    Manual</a>

<input id="files" type="file" multiple=false/>
<a class="boton" id="fileupload" href="javascript:void(0)">
Subir naturaleza</a>
</div>
```

En este fragmento de código se muestran el menú superior de la aplicación con enlaces a sitios webs con información útil para el usuario como puede ser el manual oficial. La propiedad `target="_blank"` de los enlaces permite que al pulsar sobre un enlace, este se abra en una nueva pestaña, evitando que se cierre la página.

El `input type="file"` no aparece visualmente en la aplicación, sino que es sustituido por el enlace `Subir naturaleza` para mantener una homogeneidad visual en el menú. Cuando se pulsa sobre el enlace, se ejecuta una función que acciona la entrada de ficheros.

Terminal

```
<div id="terminal"></div>
```

Dentro de esta etiqueta se cargará el emulador de terminal. El resto del código de la página carece de relevancia.

4.3. main.js

Definición de constantes

```
    const PROMPT = 'Maude> ';
    const COMANDO_NO_VALIDO = 'Comando no valido: ';
    const QUIT = 'quit';
    const entradas_no_permitidas=[
        'load',
```

```
        'cd ',
        'ls',
        'pwd'
    ]
```

Estas son las constantes utilizadas por la aplicación. El array `entradas_no_permitidas` contiene los comandos no admitidos por la aplicación por diversos motivos. El comando `quit` se encuentra aparte pues es un comando con una funcionalidad especial.

Validar comandos

```
function validar_comando(command) {
    return -1!=entradas_no_permitidas.indexOf(command);
}
```

Esta función recibe como parámetro el comando que el usuario quiere ejecutar y devuelve `true` si el comando esta dentro de `entradas_no_permitidas`, devuelve `false` en caso contrario. Huelga decir que muchos comandos Linux potencialmente peligroso que el usuario pudiera introducir son bloqueados por el propio proceso Maude, de modo que superan este filtro.

Quitar espacios y tabuladores

```
function quitar_espacios_y_tabuladores(command){
    var i=0;

    while(i<command.length&&(command[i]==' '||command[i]=='\t')){
        i++;
    }

    return command.substr(i,command.length);
}
```

El cometido de esta función no es otro que eliminar los espacios y tabuladores que el usuario pudiera haber introducido al inicio del comando con el fin de introducir algún comando no permitido. La cadena devuelta asegura que el primer carácter no es ni un espacio en blanco, ni un tabulador.

Introducir comandos

```
function introducir_comando(command, envio, socket,terminal) {
    command=quitar_espacios_y_tabuladores(command);

    if (validar_comando(command)) {
        terminal.set_prompt(PROMPT);
        terminal.error(COMANDO_NO_VALIDO + command);
    } else if (QUIT == command.substr(0, QUIT.length)) {
        location.reload();
    } else {
        socket.emit(envio, command);
    }
}
```

Esta es una de las funciones principales de la aplicación, pues se encarga de filtrar los comandos que el usuario ha introducido llamando a funciones auxiliares y enviar los comandos al servidor a través del `socket`. Primero se quitan los espacios en blanco y los tabuladores que pudiera contener la línea introducida por el usuario, a continuación se comprueba si se trata de un comando válido, en caso afirmativo se restablece el prompt y se emite un mensaje de error por la salida habilitada por el emulador de terminal para tal fin.

Si el usuario ha introducido una línea que comienza por `quit`, en consola nativa, el proceso Maude finalizaría y se volvería al terminal. Aquí se ha optado por reiniciar la consola, con lo que también se elimina el proceso Maude del servidor.

En otro caso distinto se envía el comando al servidor a través del `socket` por la vía especificada.

Función principal De acuerdo a la documentación de JQuery una función de la forma `$(function(){...})` comienza su ejecución una vez se haya cargado el documento HTML por completo. A esta función la llamaremos función principal por motivos de nomenclatura. Dentro de ella irá todo el código que requiera el documento HTML este cargado con todos sus elementos.

Emulación de click en entrada de fichero

```
$("#fileupload").click(function() {
```

```
    $("#files").click();
});
```

En este fragmentos de código se selecciona el enlace con identificador `fileupload` y se le asigna una función que se ejecutará cuando se pulse sobre él. Dicha función ejecuta un click de ratón sobre la entrada de ficheros de la aplicación, abriendo el explorador de archivos para que el usuario seleccione el fichero que desea subir a la aplicación. Gracias a esta función es posible acceder a la entrada de ficheros, pues aparece oculta en el documento HTML por medio de las propiedades CSS.

Conexión al servidor

```
var socket = io.connect('http://localhost:9090');
```

En esta línea de código es donde se realiza la conexión con el servidor y se crea el `socket` con el que se enviarán los comandos.

Entrada de comandos

```
var terminal = $('#terminal').terminal(function(command, terminal) {
    terminal.set_prompt('>');
    introducir_comando(command, 'stdin',socket,terminal);
}, {
    greetings: '',
    prompt: PROMPT,
    exit: false
});
```

En esta sección de código se crea la variable terminal con la que se gestionará el terminal que se muestra en pantalla. Se selecciona el elemento que contendrá el terminal y se le asigna la función que se ejecutará cada vez que el usuario presione la tecla `Intro`. Esta función cambia el prompt de la consola (aunque el cambio no se visualizará hasta la siguiente impresión del prompt) y se llama a la función que la enviará al servidor.

Los parámetros que se pasan a la función son `greetings`, `prompt` y `exit`. El parámetro `greetings` contendrá un texto que se mostrará al inicio de la terminal, aunque en este caso se ha dejado vacío para que no muestre nada. `prompt` contiene el texto

que se mostrará en el prompt de la terminal, este texto será cambiado con frecuencia para adoptar el comportamiento más similar a la consola real. Por último, la opción `exit` determina si finaliza su ejecución tras introducir el primer comando, por eso se le pasa el valor `false`.

Salida estándar

```
socket.on('stdout', function(data) {
    var salida = String(data);

    terminal.set_prompt(PROMPT);

    if (salida.endsWith(PROMPT)) {
        salida = salida.substr(0, salida.length - PROMPT.length);
    }

    terminal.echo(salida);
});
```

En este fragmento se muestra la función que se ejecuta cuando se recibe un evento desde el servidor `stdout`. El parámetro que recibe contiene la salida estándar producida por el proceso Maude. La función convierte a cadena de caracteres los datos, restaura el prompt a '`Maude> `', comprueba si la cadena devuelta contiene el prompt al final de la misma, en cuyo caso lo elimina de la cadena. Finalmente la imprime en la salida estándar del terminal.

Salida de error

```
socket.on('stderr', function(salida) {
    terminal.error(String(salida));
});
```

Aquí se muestra el proceder cuando se recibe un evento '`stderr`'. Su funcionamiento consiste en convertir los datos recibidos desde la salida de error en una cadena de caracteres y luego imprimirlo en la consola por medio de la salida de error de la misma.

Desconexión

```
socket.on('disconnect', function() {
    if(!alert(mensaje)){
        location.reload();
    }
});
```

En este fragmento se nos muestra la función que se ejecuta cuando se cierra la conexión desde el servidor. Muestra al usuario una alerta informándole del motivo de la desconexión. Si el usuario cierra la alerta, se fuerza una recarga de la página.

Mensaje de fin

```
socket.on('mensajeFin',function(msg){
    mensaje=msg;
});
```

Esta pequeña función es la que se ejecuta cuando llega un evento para cambiar el mensaje con el motivo del cierre de conexión. Su uso precede a un cierre de la conexión y cambiará el mensaje que el usuario vea en la alerta.

Envío de módulos

```
if (window.File && window.FileList && window.FileReader) {
    var filesInput = document.getElementById("files");
    filesInput.addEventListener("change", function(event) {
        var files = event.target.files; //FileList object
        for (var i = 0; i < files.length; i++) {
            var file = files[i];

            //Comprueba que el fichero es de texto plano
            if(file['type'] == 'text/plain'){
                var fileReader = new FileReader();
                fileReader.addEventListener("load", function(event) {
                    var comandos = event.target.result.split('\n');
                    for (var j = 0; j < comandos.length; j++) {
                        introducir_comando(comandos[j],
                            'stdin',socket,terminal);
                    }
```

```
                });
                 //Read the text file
                 fileReader.readAsText(file);
            }else{
                alert('Fichero no valido');
            }
        }
    });
} else {
    alert("Your browser does not support File API");
}
```

Este bloque de código se encarga de la subida de módulos al servidor. Primero se comprueba que la API de ficheros de HTML5 es compatible con el navegador, en caso negativo se produciría una alerta informando de este hecho al usuario al entrar en la página y no podría usar esta característica. Se selecciona la entrada de ficheros del DOM y se le agrega un escuchador de eventos para el evento 'change'. Se recogen todos los ficheros que el usuario ha seleccionado y se procesan de forma secuencial.

Para cada fichero se comprueba si es de texto plano. En caso de no ser así se informa al usuario con una alerta. Se hace uso de una funcionalidad de JavaScript para leer ficheros llamada `FileReader`. Se le agrega una función de callback para que se ejecute cuando se termine de leer el fichero. Esta función convierte todo el texto en un array cuyas casillas contienen las líneas del fichero. Para cada línea se envía al servidor por medio de la función `introducir_comando(...)`.

Una vez fuera de la función de callback, se le indica que fichero debe leer como texto.

4.4. estilo.css

```
@media screen and (min-width: 1000px){

    //Codigo CSS

}
```

Destacan el uso de las media querys, las cuales permiten cambiar las propiedades CSS de la página para adaptarlas a los distintos tamaños de pantalla. Dentro de cada

media query se coloca el código CSS que se quiere visualizar para una pantalla de esas condiciones. Deben hacer referencia a los mismos elementos, pero con distintos valores para cada propiedad CSS aplicada. Para pantallas inferiores a 1000 pixeles oculta todos los componentes, excepto el terminal, el cual se expande hasta ocupar toda la pantalla para aprovechar mejor el espacio disponible. Para pantallas igual o superiores a 1000 pixeles, se muestran todos los elementos, como el menú superior, el pie de página o un enlace al repositorio con el código del proyecto.

4.5. jquery.terminal-1.4.2.js

```
'CTRL+C': function(){
    location.reload();
    return false;
}
```

Este fragmento es la única modificación que he llevado a cabo en la biblioteca usada. Para adaptarlo a una mayor similitud con la consola nativa se ha modificado el comportamiento por defecto que ofrece la biblioteca cuando se pulsa la combinación `Ctrl+C`.

En la consola nativa, cuando se pulsa esta combinación, se elimina la ejecución del comando actual. Tras varias investigaciones, se ha determinado que este comportamiento es imposible de reproducir en este sistema, por lo que se ha optado por permitir al usuario recargar la página y con ello crear un nuevo proceso Maude con el que trabajar.

4.6. server.js

Se trata del único fichero que compone la parte del servidor. Al comienzo del fichero se declaran unas constantes útiles para el programa, como la señal que se enviará al proceso Maude, el tiempo que dura cada conexión, o el puerto por el que escuchará el servidor. Luego se importan los módulos que se usarán. El módulo express se usa para crear la base del servidor HTTP e indicar que el contenido de la carpeta `client` es estático. El módulo socket.io carga la biblioteca que se encargará de la gestión de WebSockets. Por último, el módulo `child_process` se usará para crear procesos hijos mediante la función `spawn`.

Eliminación de procesos no útiles

```
function matar_proceso_asociado(msg,shell) {
    console.log('Salida: ' + msg);

    try {
        shell.kill(senal);
    } catch (ex) {
        console.log('Excep: ' + String(ex));
    }
}
```

La función `matar_proceso_asociado(msg,shell)` recibe como parámetros un string en `msg` que contiene el motivo del fin del proceso y `shell` que representa el proceso creado. Se envía una señal `SIGKILL` al proceso Maude que mata el proceso. Si se produjese una excepción, esta sería recogida en el `catch` correspondiente y se imprimiría por consola la excepción.

Instanciación del servidor

```
app.use(express.static('client'));

server.listen(puerto, function() {
    console.log('Servidor funcionando en http://localhost:' + puerto);
});
```

Con estas líneas se instancia el servidor Node.js y se pone a escuchar en el puerto designado. La función que se le pasa como parámetro se ejecutará cuando el servidor este disponible. En este caso solo dispone de una función `console.log(...)` para informar que se encuentra listo.

La primera línea hace disponible el contenido de la carpeta `client` con una funcionalidad propia de express.js. Si no se hubiese utilizado express, la creación y configuración del servidor hubiese sido más compleja.

Usuario conectado

```
var shell = spawn('/usr/bin/maude');
```

```
setTimeout(function(){
    socket.emit('mensajeFin','Fin de sesion');
    matar_proceso_asociado('Tiempo excedido',shell);
},tiempo);
```

Una vez se ha establecido la conexión con el cliente se ejecuta una función que recibe como parámetro un `socket` con el cual transferir información con el cliente. Ese parámetro `socket` será el mismo en todas las descripciones venideras.

Se crea una variable `shell` con la cual se interactuará con el proceso Maude creado por la función `spawn`. Esta función hace uso del módulo `chil_process` para crear un proceso hijo.

La función `setTimeout(...)` de JavaScript ejecuta un trozo de código una vez se ha superado un periodo de tiempo especificado como parámetro. En este caso se emite a través del `socket` el mensaje que recibirá el usuario cuando finalice la sesión y se llama a la función encargada de eliminar el proceso Maude asociado a la conexión.

Fin del proceso Maude

```
shell.on('exit', function(code, signal) {
    console.log('exit');
    socket.disconnect();
});
```

Esta función se ejecuta cuando finaliza el proceso Maude y recibe como parámetro el código de finalización del programa y la señal que lo provocó si la hubiera. El código llama a la función de la biblioteca `socket.io` encargada de desconectar el `socket`.

Salida estándar

```
shell['stdout'].setEncoding('ascii');
shell['stdout'].on('data', function(data) {
    socket.emit('stdout', data);
});
```

Este fragmento se ejecuta cuando el proceso Maude emite una salida estándar. Esta es reenviada al cliente mediante el `socket`. La primera línea solo se ejecuta una vez

en cada conexión y asigna una codificación `ASCII` al flujo de datos proveniente de la salida estándar del proceso.

Salida de error

```
shell['stderr'].setEncoding('ascii');
shell['stderr'].on('data', function(data) {
    socket.emit('stderr', data);
});
```

Análogamente al bloque anterior, la primera instrucción se encarga de darle una codificación `ASCII` a la salida de error. El resto del código se encarga de trasmitir al cliente toda salida de error que emita el proceso Maude por una vía distinta a la salida estándar.

Entrada estándar

```
socket.on('stdin', function(command) {
    shell.stdin.write(command + '\n') || socket.emit('disable');
});
```

Esta función se ejecuta cuando se produce un evento `stdin`. Este evento sucede cada vez que desde el cliente se envía un comando. La función recibe como parámetro el comando que el usuario ha introducido (sin espacios ni tabuladores al inicio). Escribe el comando en la entrada estándar del proceso y concatena un salto de línea para introducirlo en el proceso.

Desconexión

```
socket.on('disconnect', function() {
    matar_proceso_asociado('Desconexion',shell);
});
```

Cuando se produce un evento de desconexión desde el cliente se ejecuta este fragmento de código, en el cual se llama a la función encargada de eliminar el proceso Maude asociado a la conexión.

4.6. server.js

Capítulo 5

Conclusiones

El objetivo del presente trabajo de fin de grado es la creación de un entorno online en el que poder ejecutar comandos del lenguaje Maude. El objetivo se ha logrado satisfactoriamente, aún con las pocas limitaciones que ofrece el sistema desarrollado respecto a la consola nativa.

Versiones posteriores de este software podrían estudiar como implementar un sistema que limite individualmente el tiempo de ejecución de un comando, en lugar de limitar el tiempo de conexión. Otra mejora de gran interés sería la implementación de otra biblioteca para emular el terminal que funcione en todo tipo de plataformas, ya sean de escritorio o dispositivos móviles.

El código se ha desarrollado de la forma más modular posible de forma que sea fácilmente reutilizable en caso de querer implementar una consola online para otros lenguajes de programación cambiando unos pocos parámetros y constantes.

Bibliografía

[McF12] David Sawyer McFarland. *JavaScript y jQuery*. Anaya, 2012. (No citado)

[nod] Página oficial de Node.js. (No citado)

[Orn13] George Ornbo. *Node.js*. Anaya, 2013. (No citado)

[ter] Página oficial del emulador de terminal. (No citado)

[TR16] Manuel Ángel Torres Remon. *Diseño web con HTML5 y CSS3*. MARCOMBO, 2016. (No citado)

[web] Tutorial sobre websockets. (No citado)

[09] José María Álvarez Palomo. Apuntes de la asignatura tipos abstractos de datos. 2009. (No citado)

Bibliografía

Manual de usuario

Una vez accedido a la página Terminal Maude se tendrá disponible la total funcionalidad de la aplicación durante una hora. El control de la aplicación es idéntico a una consola nativa, por lo que si se está acostumbrado a su utilización, el uso de la aplicación será trivial para el usuario.

Si el terminal se encuentra seleccionado, aparecerá el cursor parpadeando y entonces podrá escribir sobre ella. Si por el contrario no visualizase el cursor parpadeante, haga click sobre el terminal para que aparezca. Teclee la instrucción que desee ejecutar.

Puede utilizar las teclas de flecha izquierda y derecha para mover el cursor sobre la línea actual. Siempre que pulse una tecla, la acción asignada se ejecutará sobre el carácter situado a la izquierda del cursor.

Si desea borrar un carácter de la línea actual, sitúe el cursor después del carácter y pulse la tecla de retroceso de su teclado.

Una vez la línea actual sea de su agrado, pulse la tecla `Intro` para ejecutar el comando actual. Una vez se haya resuelto, su salida se mostrará en la pantalla. Si esta apareciera en rojo, se debe a que ha habido algún tipo de error, por favor lea el mensaje.

Si desea volver a introducir un comando que ha escrito previamente, no es necesario que vuelva a escribirlo. Pulse la tecla de flecha hacia arriba para navegar por el historial de comandos y ver todos los que ha introducido previamente. Si desea volver a un comando posterior en la navegación, basta con que pulse la tecla de flecha hacia abajo. Una vez seleccionado el comando que desee solo pulse la tecla `Intro` para volver a ejecutarlo.

Si desea reiniciar la consola solo tiene que recargar la página o con el terminal seleccionado pulsar la combinación `Ctrl+C`

Si introduce una serie de comando con mucha frecuencia, tal vez le interese subir

una naturaleza. Cree un fichero de texto plano y escriba en cada línea un comando diferente, luego guardelo en un lugar accesible para el navegador. Seleccione la opción "Subir naturaleza"situada en la parte superior de la página. Seleccione el fichero que ha creado y pulse aceptar. Esto ejecutará cada línea del fichero con el interprete de Maude.

Si desea obtener mas información sobre el lenguaje Maude pulse sobre alguno de los enlaces disponibles en la parte superior de la página, donde encontrará enlaces al manual o la página oficial, entre otros.

Manual de instalación

Primero debemos usar una distribución Linux para nuestro servidor. En nuestro caso usaremos Ubuntu 16.04 LTS. Nos conectaremos por el comando `ssh` a nuestro servidor. Una vez conectados, instalaremos los paquetes necesarios.

Instalaremos Node.js en nuestro sistema. Para ello accedemos a su página oficial Página oficial de Node.js y seguimos las instrucciones para nuestra distribución. Este software ha sido desarrollado y probado con la versión 6 LTS de Node.js. Seguidamente instalamos Maude. En el caso de Ubuntu se ejecuta el comando `apt install maude`. Para otras distribuciones consulte como instalar paquetes.

Descargamos el código en el servidor desde el repositorio. Accedemos a la carpeta raíz del proyecto y ejecutamos el comando `npm install` para instalar las dependencias. Para arrancar el programa ejecutamos el comando `npm start`. Y Ya tenemos nuestro servidor en marcha y disponible.